#10
1-24-03
DW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Applicant: David K. Vavro et al.	§	Art Unit:	2183
Serial No.: 09/465,634	§	Examiner:	Tonia Moenske
Filed: December 17, 1999	§		
For: Digital Signal Processor Having A Plurality Of Independent Dedicated Processors	§	Atty Docket:	ITL.0286US P7814

Board of Patent Appeals & Interferences
Commissioner for Patents
Washington, D.C. 20231

APPEAL BRIEF

Sir:

Applicants respectfully appeal from the final rejection mailed October 29, 2002.

I. REAL PARTY IN INTEREST

The real party in interest is the assignee Intel Corporation.

II. RELATED APPEALS AND INTERFERENCES

None.

III. STATUS OF THE CLAIMS

Claims 1-24 are rejected. Each rejection is appealed.

Date of Deposit: January 15, 2003
I hereby certify under 37 CFR 1.8(a) that this correspondence is being deposited with the United States Postal Service as **first class mail** with sufficient postage on the date indicated above and is addressed to the Commissioner for Patents, Washington DC 20231.

Cynthia L. Hayden

Cynthia L. Hayden

IV. STATUS OF AMENDMENTS

All amendments were entered.

V. SUMMARY OF THE INVENTION

A digital signal processor 10 may include a plurality of microprocessors 14, 18, 20, 24 and 26 each having their own instruction sets. The individual processors need not communicate directly with one another but instead may communicate through storage registers associated with a general purpose register (GPR) 32 that is part of the registers 16. Thus, the results of an operation performed by one of the processors may be stored in the GPR 32 for access by another processor.

Each of the processors may be separately programmed with its own set of codes. The instruction sets for each processor may provide the logic for operating the particular functions for that processor, avoiding the need for separate hardware logic for implementing the subprocessor functions, in one embodiment of the invention.

The master programmable controller (MPC) 18 provides the timing for the other processors and operates like an instruction execution controller. Knowing the times to execute a given instruction in a given processor, the MPC 18 waits for response from a given processor. In effect then the MPC 18 has instruction sets that enable it to assist others to operate on a cycle by cycle basis. Generally, one instruction is executed per cycle.

Although each of the processors may be independently programmed, the instruction sets may be sufficiently similar so that an instruction set for one processor may be modified for use in other processors. This may decrease the time for programming each processor. See specification at page 3, line 16 through page 4, line 12.

A programmable input processor (PIP) 14 receives inputs from a receive buffer such as a first in first out (FIFO) register 12. The PIP 14 may, in some embodiments of the present invention, provide a precision change or a scaling of an input signal. The PIP 14 may be advantageous since it may provide for input data processing when input signals are available and may provide mathematical operations at other times. The input data need not be synchronized with the system 10 since the MPC 18 may wait for the PIP 14 to complete a given operation. Thus, in effect, the MPC 18 provides the synchronization for a variety of unsynchronized subprocessor units.

The programmable output processor (POP) 20 provides outputs to a transmit buffer 22 such as a first in first out (FIFO) register. The POP 20 may also do mathematical operations when no output data is available for transmission to the transmit buffer 22.

The programmable random access memory (RAM) processor (PRP) 20 basically stores and retrieves data. It may be particularly advantageous in storing intermediate results, in a cache-like fashion. This may be particularly applicable in two-dimensional image processing.

Some embodiments of the present invention may use normal length words but other embodiments may use the so-called very long instruction word (VLIW). VLIW may be advantageous in some embodiments because the logic may be in effect contained within the instructions and glue logic to coordinate the various subprocessors may be unnecessary. See specification at page 4, line 13 through page 5, line 9.

Since each instruction may have a predetermined execution time independent of the data, the MPC 18 can control the various processor operations, on a cycle-by-cycle basis. Each of the processors is capable of operating in parallel with all of the other processors. Thus, in effect, the

architecture shown in Figure 1 is a parallel processor; however, the architecture is such that the operations are largely broken down on general recurring functional bases.

A number of mathematical processors may be provided within the unit 26 based on the particular needs in particular applications. In the illustrated embodiment, a pair of identical add and subtract programmable mathematical processors (PMP) 28a and 28b are combined with a pair of multiply and accumulate (MAC) programmable mathematical processors (PMP) 30a and 30b. However, a variety of other mathematical processors may be plugged into the digital signal processor 10 in addition to or in place of any of them or all of the illustrated PMPs.

Each of the processors 14, 18, 20, 24, 28 and 30 may be programmable, contain its own random access memory and decode the random access memory contents into instructions, in one embodiment of the invention. The control of the programmable processors is accomplished by the MPC 18. It controls when a given instruction is executed by any of the programmable processors.

Thus, the MPC 18 controls the time of execution of instructions and is the only provider of instructions that are clock cycle active. The remaining programmable processors run at the same time.

The register module 16 contains general purpose registers for storing data and allowing the accessing of data by the programmable processors. The inclusion of a programmable random access memory processor 24, programmable input processor 14 and the programmable output processor 20 allows very flexible input, output and data manipulation/storage operations to take place in parallel with mathematical operations. See specification at page 5, line 10 through page 6, line 12.

MPC 18

The MPC 18 controls the processors 14, 20, 24, 28 and 30 that may be considered as slave processors to the MPC 18. Thus, the MPC 18 contains an instruction memory 40 and an instruction decode 38, as indicated in Figure 2. The MPC 18 determines when a slave processor can execute instructions and the slave processors communicate when data reads or writes to the external data bus 19 have completed. The MPC 18 is also responsible for generating cycle dependent or instruction dependent signals. Examples of such signals include interrupts, data tags and the like.

In some embodiments of the present invention, the MPC 18 may be the only processor concerned with the slave processor timing. The MPC 18 may have the ability to meter control signals and globally affect the slave processors based on the state of those metered signals. The MPC 18 may also control when the processors execute a given instruction. This module is clock cycle accurate and may be used to control the parallel operation of an embodiment using VLIW. The MPC 18 contains instruction enables for each of the other processors. These instruction enables are used to control when the slave processor processes its next instruction. Null operations are performed by not issuing an enable during a particular clock cycle.

The MPC 18 provides a single central master processor to control the operation of the slave processors. This allows easier portability across different process technologies used to make the various processors. As the slave processors are added, removed, modified or redesigned to meet timing with different throughputs, only the master processor program may change. This avoids the need to completely redesign the entire digital signal processor 10. With an instruction assembler that generates the machine code for the master control processor, this process may be relatively fast and easy. Easy generation of cycle accurate or program dependent

signals may be accomplished with the MPC 18. Slave processors may be removed or added with ease allowing the creation of custom digital signal processors with different performances.

PIP 14

As shown in Figure 3, the PIP 14 includes an instruction memory 46, an instruction decoder 44 and a math capability 48. The PIP 14 is capable of implementing addition, subtraction and shift left functions on the input data as well as internal data in accordance with one embodiment of the present invention. The MPC 18 controls the execution of instructions by the PIP 14. The PIP 14 signals the MPC 18 when input data reads are complete. The PIP 14 uses self-timed math modules to execute instructions and math functions. See specification at page 9, line 15 through page 10, line 14.

The PIP 14 may send incoming data from a receive FIFO register 12 into the GPR 32. The PIP 14 has the ability to add a full signed 16-bit offset and scale up by shifting left the incoming or internal data. The PIP 14 also has overflow and underflow error flags that can be used by other entities to determine what to do with the data. When operating on internal data, the PIP 14 may read or write data from any GPR 32 register and this mode is clock cycle accurate.

When operating on data from the input FIFO register 12, the PIP 14 may write data to any GPR 32 register. The PIP 14 interfaces with the receive FIFO register 12 with a busy/valid protocol. An instruction done signal may be sent to the MPC 18 whenever the receive FIFO register 12 instruction is completed.

The PIP 14 may also set up register chains in the GPR 32. For example, in the chain mode, data written into a register zero does not destroy data in the register zero. Instead the data from the register zero is automatically written to a register one and data from the register one is

automatically written to the register two and so on until the end of chain (EOC) is reached. Thus, the register zero is now defined as the start of chain (SOC) because the PIP 14 is writing to register zero. This may all happen in one clock cycle, allowing fast sliding filter operations in both one and two dimensions. See specification at page 10, line 15 through page 11, line 12.

Through the use of the PIP 14, data may be transferred from working registers in a programmable fashion. Data input and other slave operations may occur independently of each other. The PIP 14 allows writing data to a destination that may not be ready for data. While not performing data transfers, the PIP 14 can be used to perform internal math functions. See specification at page 12, line 10 through page 13, line 15.

POP 20

The POP 20 has an instruction memory 54, an instruction decode 52 and a math unit 56 as shown in Figure 4. The POP 20 is capable of implementing addition, subtraction, shift right, ceiling, flooring, absolute value and round functions on internal data in accordance with one embodiment of the present invention. The POP 20 also has overflow and underflow flags. The MPC 18 controls the execution of instructions by the POP 20. The POP 20 signals to the MPC 18 when output data writes are complete. The POP 20 uses self-timed math modules to execute instructions and math functions.

The POP 20 is responsible for sending data to the transmit buffer 22 from any of the GPR 16 registers while in the transmit data mode of operation. While in the internal mode, the POP 20 may send and receive data from any of the GPR 16 registers. The internal mode of operation occurs when the POP is not sending data to the transmit FIFO register 22.

The transmit data mode of operation may not be clock cycle accurate where the internal mode of operation may be clock cycle accurate. Transfers to the transmit FIFO register 22 may

not be clock cycle accurate because the register 22 may be full. However, the POP 20 can signal the MPC 18 when an instruction is finished being executed. See specification at page 13, line 16 through page 14, line 8.

With the POP 20, data may be transferred from working registers in a programmable fashion. Data output and other slave operations may occur independently of one another. The POP 20 also allows the data to be written to a destination that may not be ready for the data. When not performing data transfers, the POP 20 can be used to perform internal math functions. See specification at page 15, line 9 through page 16, line 13.

Registers 16

The registers 16 include a bus interface 34 and N general purpose registers 32 configured to allow the chaining and global chaining modes as well as independent read and write operations that can occur from a number of processors at the same time. The GPR 32 allows independent data transfers from and to any of the other processors. The GPR 32 includes registers for each of the processors that can be written to by any processor module. If two processor modules try to write to the same register, an error flag is set.

In chaining mode, the GPR 32 may be configured to chain one register's output to that of another register. The data written to a register 0 from the PIP 14 does not destroy data in the register 0, but instead the data from the register 0 is automatically written over to register 1 and data from the register 1 is automatically written over to the register 2 and so on in one clock cycle until a programmable EOC is reached. Thus, as shown in Fig. 5, when data is written into GPR zero indicated at 58, the data is automatically transferred to GPR one indicated as 60 and so on.

The SOC is defined as the present GPR location that the PIP 14 is writing to. This allows fast Finite Impulse Response (FIR) filters as well as fast sliding filter operations in both one and two dimensions. As an example, if SOC is set to six, a write from the PIP 14 to GPR six in this example produces no chaining. Allowing only the PIP 14 to define the SOC allows fetching of the next set of data in order while the last set is being operated on without using register to register move instructions. When contention occurs, any writes to any GPR by a processor take precedence over chaining rights from the previous register. See specification at page 16, line 14 through page 17, line 11.

The use of a global data chaining, shown in Fig. 6, allows data to be processed more efficiently when implementing IIR filters. Global data chaining is defined as allowing internal math modules to form the SOC. This allows computed data to generate an SOC as opposed to allowing only the PIP 14 to form a data chain. In the global chaining mode, the PIP 14 cannot define the SOC. If the global chaining mode is active, then any write to a GPR from any processor except the PIP 14 can define a valid SOC. When performing IIR filtering, the SOC may be defined by other processors because the input data may be operated on before insertion into the chain. When contention occurs, any writes to the GPR by a processor take precedence over chaining rights from the previous register.

Referring to Figure 13, chaining code 106, which may be stored in association with the MPC 18, begins by determining whether global chaining has been selected as indicated at diamond 107. If global chaining has been selected, a check determines whether the PIP 14 has written to a general purpose register 0 as indicated in diamond 108. If so, GPR 0 is set equal to the new word and no chaining is indicated. If the PIP did not write to GPR 0, a check at diamond 112 determines whether there is any other processor write to register GPR 0. If so, the

SOC is set equal to zero and the EOC is set equal to a programmable value greater than or equal to the SOC. GPR 0 is set equal to the new word and GPR 1 is set equal to GPR 0, and GPR 2 is set equal to GPR 1, and GPR (EOC) is set equal to GPR (EOC-1) as indicated in block 114. If GPR (EOC) equals GPR 0, then no chaining takes place. A write to a GPR always occurs. See specification at page 17, line 12 through page 18, line 8.

If global chaining has not been selected, then a check at diamond 116 determines whether the PIP 14 has written to a GPR register X (GPR(X)). If not, a check at diamond 120 determines whether there are any other writes to the register X. If so, the register X is set equal to the new word and no chaining occurs, as indicated in block 122.

If the PIP did write to the register GPR (X) then the SOC is set equal to the GRP(X) as indicated in block 118. The EOC is set equal to a programmable value greater than or equal to the SOC. The GPR (X) is set equal to the new word and the register GPR (X+1) is set equal to GPR (X), GPR (X+2) is set equal to GPR (X+1) and GPR (EOC) is set equal to the register GPR (EOC-1). If the register GPR (EOC) is less than or equal to the register GPR (X), then no chaining takes place but a write to GPR (X) always occurs.

The shift from one register to another may take place in one clock cycle. For example, the shift from register one to register zero, the shift from register two to register one and the shift from register three to register two and so on, all may occur in one clock cycle, in one embodiment of the present invention.

Use of chaining and global chaining modes allows independent data processing to occur in any of the processors. In addition it may allow faster IIR filters, FIR filters, sliding N dimensional filters and vector products, without the need for a large number of register to register instructions.

PRP 24

The PRP 24 includes a number of random access memory (RAM) modules 74, as shown in Figure 7. The number of modules 74 equals the number of sub-processors that use the PRP 24. Thus, the N RAMs 74 are coupled to an instruction decode unit 72 which in turn is coupled to an instruction RAM 76. The N RAMs 74 can be programmed to read and write based on instructions contained in the instruction memory 76. Each RAM 74 is able to read and write independently of the others. See specification at page 18, line 9 through page 19, line 10.

The PRP 24 may allow internal storage of N 16 bit data blocks in one embodiment of the invention. The PRP 24 may be used for filter operations where data is used recursively or data flow would be too restrictive on performance. An example is performing two dimensional discrete cosine transforms (DCT) where the filtering is performed on eight columns and then on eight rows. Another example is in direct storage of quantization tables and memory so that the zigzag operations and quantization may take place at the same time.

The PRP 24 may read or write to any GPR 16 register. Since the PRP 24 contains N separate memories 74, N reads or writes or a mix may take place at the same time. The firmware has direct access to all of the N RAM memories 74.

The PRP 24 allows coefficients and data to be saved in a local RAM so as to be available for data processing without reading data or coefficients from an external device. This may reduce input/output performance degradation during signal processing.

The use of a RAM to store a large number of coefficients results in the use of a smaller area than using registers. All RAM reads and writes are controlled by instructions and can operate independently of other slave processor operations, in one embodiment of the invention. This may allow faster signal processing.

PMPs 28, 30

The PMPs 28 include an instruction decode unit 80, an adder/subtractor 84 and an instruction RAM 82 as shown in Figure 8. A PMP 28 performs addition or subtraction of two inputs and sends the result to the GPR 32. The source and destinations are defined by the instruction. The processor executes the instructions using self-timed mathematical modules. See specification at page 20, line 5 through page 21, line 11.

The main function supported by a PMP 28, in one embodiment of the invention, is to add or subtract two signed 16 bit numbers and output a 16 bit signed result. The PMP 28 also has overflow and underflow flags. The PMP 28 can receive data from any of the GPR 32 registers and can provide data to any of the GPR 32 registers.

The use of a PMP 28 allows addition and subtraction operations to occur independently of other processors. Since the PMP 28 is fully modular in design, it allows scalablility of the overall digital signal processor 10. See specification at page 21, line 12 through page 22, line 6.

The PMP 30 is a multiply and accumulate (MAC) processor with its own instruction memory 90, instruction decode 88, and math module 92 as shown in Figure 9. The processor performs multiply and accumulate operations, and sends the results of its operations to a GPR 32 register. The source and destination are defined by the instruction. The processor may execute instructions using self-timed mathematical modules.

The main supported function of the PMP 30 is to multiply two signed 16 bit numbers to produce a 32 bit result. The result may be added from a previous 32 bit result to form a multiply and accumulate (MAC) function. The accumulator size is 32+N bits allowing for internal extended precision operation. The results of the accumulator are rounded to 16 bits and shifted

right 16 bits to produce a signed 16 bit result. The PMP 30 also has overflow and underflow flags. The PMP 30 may receive data from any GPR 32 and may provide data to any GPR 32.

The PMP 30 allows multiply or multiply and accumulate operations to incur independently of other processors. The processor is fully modular allowing scalability of the digital signal processor 10.

Data Path Interface

Referring to Fig. 10, a data path interface method used in the digital signal processor 10, allows self-timed single cycle or multi-cycle math processors to be interchanged without affecting the instruction decode. This enables greater portability of the processor 10 to different process technologies. The interface also allows the implementation of self-timed instructions that are dependent only on math processor timing delays.

All data inputs to the arithmetic elements 94 (such as a PMP 28 or 30) are registered outputs and stable until other data values are presented. A valid input signal (IN_VALID) is provided when new data is supplied. New data is only supplied if a busy signal from an arithmetic element 94 is not asserted.

In a pipeline element 96, shown in Figure 11, the busy signal may always be low because data can be continuously fed to the arithmetic element 96. A delay element 100 may be used to delay the operation of the arithmetic element by as much as one clock cycle. For example, the math processor may be divided into units 102 and 104 with a delay element 100 in between. Similarly, the input valid signal (IN_VALID) may be delayed by one cycle delaying the output valid (OUT_VALID) by one cycle. Likewise, the data direction signal (IN_DEST_ADDR), which tells where the data may go, may be delayed. See specification at page 23, line 12 through page 24, line 14.

In a multi-cycle arithmetic element 98, shown in Figure 12, the busy signal can be used to hold off new data from being sourced to the arithmetic element 98. Destination addresses for the result of the mathematical operation and mode change signals may be supplied to the arithmetic element 98 to help stabilize it until new data is present.

The arithmetic element 98 provides internal delays to match the latency of the arithmetic such that multi-cycle operation occurs. The operation may be spread over two or more clock cycles. The input data valid (IN_VALID) and input destination (IN_DEST_ADDR) signals may also be delayed the needed number of cycles (N). Error flag signals are provided and registered by the arithmetic element. The input data valid signal to qualify the input data, and the mode or control signals to the arithmetic element 98 are asserted for new data sent to the arithmetic element.

These interface methods allow a digital signal processor to transcend different process technologies. In some cases the sole redesign needed for a new function may be to redesign the math modules. Instructions set in instruction decode logic need not be changed to accommodate different arithmetic element timing changes. This allows a more portable design amenable to different process technology changes.

Through the use of pipelined or multi-cycled processes, different mathematical processors may be added to the overall processor regardless of whether they require more or less time than the processor which they replace. Thus, in cases where a slower math processor is replacing a faster math processor, a pipelined or a multi-cycled architecture may be utilized to compensate for the additional delay time. Conversely, if the new math processor is faster than the one which it replaces, the faster math processor may be used without change except as described hereinafter. See specification at page 24, line 15 through page 25, line 16.

In each case, the MPC 18 is recompiled to adjust to the slower or faster timing of a new math processor. Regardless of whether the new timing is longer or shorter, all that is needed is to recompile the MPC 18. The MPC 18 then operates with the new timing. Thus, the system may be easily and quickly adapted to new processors which are made on different process technologies and which may be faster or slower than the processor for which the system was originally designed. See specification at page 25, lines 17 through 23.

VI. ISSUES

A. Is Claim 15 Indefinite?

B. Is Claim 1 Anticipated by Kitamura?

C. Is Claim 16 Anticipated by Kitamura?

VII. GROUPING OF THE CLAIMS

Claims 1-15 may be grouped for consideration of the prior art rejection.

Claims 16-24 may be grouped.

VIII. ARGUMENT

A. Is Claim 15 Indefinite?

Claim 15 was rejected on the grounds that the word "multi-cycled" is indefinite. With respect to the objection to the term "multi-cycled" mathematical processor, a multi-cycle element is explained on page 24, line 15 through page 25, line 16, (and as explained in the material set forth in the summary herein at page 14, last paragraph through the end).

As explained on page 14 infra, the use of multi-cycled elements allows interchangeability without affecting the instruction decode. A busy signal can be used to hold off new data from

being sourced to the ineffected unit. Internal delays may be used to match latency such that multi-cycle operation occurs. This allows the various processors in the overall system to be exchanged and changed out regardless of whether they may involve operations spread over two or more clock cycles. For example, input data valid and input destination signals may be delayed that needed number of cycles to make different processors working on different cycles coherent.

Different multi-cycled mathematical processors may be added to the overall process regardless of whether they require more or less time than the processor which they replace. Thus, in cases where a slower math processor is replacing a faster math processor, a multi-cycled architecture may be utilized to compensate for the additional delay time. The MPC 18 is recompiled when a new processor that is slower or faster is added to adjust to the slower or faster timer of the new processor.

Therefore, the Section 112 rejection of claim 15 should be reversed.

B. Is Claim 1 Anticipated by Kitamura?

Claim 1 was rejected under Section 102 as being anticipated by Kitamura. It was indicated that Kitamura teaches a mathematical processor, namely element 5, in Figure 6.

However, as indicated in column 11 of Kitamura, paragraph 39, beginning at line 47, the data processing unit 5 splices the video data DA and DB in response to a splicing instruction from CPU 7. There is no indication that this device can be considered to be a mathematical processor. Thus, at least one element is missing in Kitamura and, therefore, the Section 102 rejection should be reversed.

With respect to the argument that, in essence, "mathematical processor" simply means any processor, it is clear that this cannot be so since mathematical processor is a well known

term of art. Clearly mathematical processors are processors devoted to doing mathematical operations such as add and subtract. See, e.g., page 5, lines 17-24. This understanding is further supported by the attached academic paper.

Therefore, the rejection of claim 1 should be reversed.

C. Is Claim 16 Anticipated by Kitamura?

Similarly, claim 16 calls for using a third processor for mathematical operations. No such thing is shown in Kitamura.

IX. CONCLUSION

Applicants respectfully request that each of the final rejections be reversed and that the claims subject to this Appeal be allowed to issue.

Respectfully submitted,

Date: 11/15/03

Timothy N. Trop, Reg. No. 28,994
TROP, PRUNER & HU, P.C.
8554 Katy Freeway, Ste. 100
Houston, TX 77024
713/468-8880 [Phone]
713/468-8883 [Fax]

APPENDIX OF CLAIMS

The claims on appeal are:

1. A digital signal processor comprising:

a mathematical processor;

an input processor that processes input signals to the digital signal processor;

an output processor that processes output signals from the digital signal processor;

a master processor that controls said mathematical processor, said input processor and said output processor; and

a storage selectively accessible by each of said processors.

2. The digital signal processor of claim 1 further including a random access memory processor that stores intermediate calculation results.

3. The digital signal processor of claim 2 including a bus coupling each of said processors to said storage.

4. The digital signal processor of claim 1 wherein said input and output processors also implement mathematical operations.

5. The digital signal processor of claim 1 wherein each of said processors have their own instructions sets.

6. The digital signal processor of claim 1 wherein said processors communicate with one another through said storage.

7. The digital signal processor of claim 1 wherein each of said processors use very long instruction words.

8. The digital signal processor of claim 1 wherein said master processor provides the timing for the other processors.

9. The digital signal processor of claim 1 wherein said master processor waits for the input processor to complete a given operation.

10. The digital signal processor of claim 1 wherein each of said processors includes its own random access memory.

11. The digital signal processor of claim 1 wherein said storage includes a plurality of registers, said registers automatically transfer existing data from a first register to a second register when new data is being written into said first register.

12. The digital signal processor of claim 11 wherein said input processor causes the automatic transfer of data.

13. The digital signal processor of claim 11 wherein said mathematical processor causes said data to be transferred from one register to another.

14. The digital signal processor of claim 1 including a mathematical processor which is pipelined.

15. The digital signal process of claim 1 wherein said mathematical processor is a multi-cycled mathematical processor.

16. A method of digital signal processing comprising:

using a first processor to process input signals to said digital signal processor;

using a second processor to process output signals from said signal digital signal processor;

using a third processor for mathematical operations;

controlling said first, second and third processors using a fourth processor; and

enabling each of said processors to selectively access a storage.

17. The method of claim 16 including providing the timing from said fourth processor for each of the other processors.

18. The method of claim 16 including automatically transferring data from a first register in said storage to a second register in said storage when new data is being written into said first register.

19. The method of claim 18 including automatically transferring said data in response to action by said first processor.

20. The method of claim 18 including automatically transferring said data in response to action by said third processor.

21. The method of claim 18 including storing a bit which indicates which processor may control said automatic transfer of data from one register to another.

22. The method of claim 16 including accommodating for timing differences between said processors by operating one of said processor in a pipelined fashion.

23. The method of claim 16 including accommodating differences in processing cycle time of one of said processors by operating said processor in a multi-cycle mode.

24. The method of claim 23 including holding off said fourth processor when one of said processors is taking more than a cycle to complete an instruction.

VLSI Architecture for Pre-computation of Rotation Bits in Unidirectional Flat-CORDIC

Satish Ravichandran and Vijayan K. Asari
Department of Electrical and Computer Engineering
Old Dominion University, Norfolk, VA 23529

Abstract – *CORDIC is an iterative algorithm to compute the trigonometric, logarithmic and transcendental functions for digital signal processing applications. Flat-CORDIC unravels the basic add and shift operations in CORDIC and uses pre-computed values of direction of rotation to perform the computations simultaneously. A new algorithm for the implementation of CORDIC with unidirectional vector rotation is proposed in this paper, which reduces the number of iterations significantly. A novel architecture for pre-computing the rotation bits for flat-CORDIC implementation is also presented. The architectures are being implemented using Altera Quartus II for programming the Apex II FPGA chip, and the results are encouraging.*

1. Introduction

The digital signal processing techniques have been used in many fields today from Internet to wireless phones. DSP requires complex mathematical computations and there is a need for a high-speed mathematical processor for an efficient signal processing application. CORDIC is an acronym for COordinate Rotation Digital Computer was first introduced by Volder [1] in 1959 and later was bought into light again by Walther [2] in 1971. CORDIC is an iterative hardware efficient algorithm that does high-speed mathematical operation in linear, circular and hyperbolic coordinate system. CORDIC architectures find its use in many digital signal and image processing [3][6], digital filter design, matrix inversion, Eigen value computation, SVD algorithms [10] etc. This paper attempts to develop a novel method for fast and area efficient CORDIC architecture.

The architecture design for CORDIC has been undergoing various modifications to improve the performance as well as the VLSI area required for its implementation. The number of iterations is halved by using a radix-4 architecture [4] but the complexity of the hardware is increased. Carry-save additions [5] are incorporated in the iterations thereby reducing the complexity of the hardware. Other architectures and algorithms such as Backward Angle Recording [7] algorithm, Hybrid algorithms [8] and pipeline architecture have been used to reduce the number of iterations and the hardware used. Most of the architectures discussed reduce the number of iterations by using a constant scaling factor for multiplication. One proposed architecture to improve the performance is unrolled or flat CORDIC architecture where, each iteration is done independently and concurrently, thereby reducing the time of operation but increasing the hardware real estate.

Flat CORDIC involves unrolling the iterations and executing the iteration in parallel. To build this architecture, the sign of rotation (clockwise or counterclockwise) is pre-computed and these signed bits are used to generate the angle. Therefore, generation of signed bits play a very important and crucial role in the performance of the CORDIC architecture. One of the

of using the flat architecture is that the speed of operation of the device would be increased at the cost of the VLSI area needed to implement the CORDIC algorithm. Since, the architecture is highly parallel, the amount of space used in building this architecture is more compared to other conventional architectures. In this paper, a technique is proposed to pre compute the signed digits and also optimize the speed, area and power of the architecture. The conventional CORDIC rotates in two directional (clockwise or counterclockwise). A new algorithm is proposed which uses only unidirectional rotation. Therefore the value to be measured is obtained by iterating through only counter-clockwise direction and skipping the

angle in the other direction. Thus, the number of iterations for any given value is reduced significantly depending on the input value.

2. The CORDIC Algorithm

The algorithm of CORDIC proposed by Volder, is derived from the two rotation equations [9],

$$x' = x\cos\phi - y\sin\phi \quad (1a)$$

$$y' = y\cos\phi + x\sin\phi \quad (1b)$$

The equations represent a vector (x, y) that is rotated by an angle ϕ. This equation is modified such that the rotations are characterized by an iterative shift operation in digital logic. Therefore, the modified equation is given by,

$$x_{i+1} = x_i - y_i .m.d_i.2^{-i} \quad (2a)$$

$$y_{i+1} = y_i + x_i .m.d_i.2^{-i} \quad (2b)$$

where m = -1, 0, 1 and it represents the rotation in the specific coordinate system (m=0 for linear, m = 1 for circular and m = -1 for hyperbolic system) and $d_i = \pm 1$ represents the sign of rotation ($d_i = 1$ for counterclockwise rotation and $d_i = -1$ for clockwise rotation).

As discussed above, in Flat CORDIC architecture, the angle is fed as the input and the signs of rotations are to be pre-computed from the given angle. A mathematical treatment of pre-computing the angles for any given angle is presented in [11]. It can be seen that we need to compute only the first (N/3 –1) bits of the angle and the signed bits are taken directly from the remaining bits. Initially, all the values of the signed bits corresponding to the first (N/3 –1) digits are pre-stored in a ROM.

3. Implementation of the unidirectional CORDIC Algorithm

The basic idea behind the proposed CORDIC algorithm is that the rotation is done only in the counter-clock direction. Initially, the angle whose corresponding trigonometric values are to be computed is stored in a register. As explained before, the signed bits for the first (N/3–1) bits are calculated separately, as these bits dominate in calculating the accuracy of any given angle. The LSB (Least Significant Bits) (N/3–1) to N bits is found out from the remainder of the angle obtained after the first (N/3–1) bits are calculated. The corresponding value of the remainder found is added to the remaining angle from the input angle (N/3 – N). The algorithm used in the calculation of the first (N/3–1) bits is given below. The angles corresponding to the (N/3 – 1) bits are calculated and loaded in the registers.

Algorithm:

Step 1: Take the first (N/3 — 1) bits from the input and is stored in register. The remaining bits are stored in another register REM.

Step 2: The first of the (N/3 –1) bit is checked for a 0 or 1. If it's a 1, then sign bit [0] = 1 and the REM is fed back with the value of REM – the stored angle for the corresponding bit position. If the bit in the REM is a '0' then sign bit [0] = 0 and REM value is not changed.

Step 3: Check whether (N/3 – 1) bits are done. If yes goto Step 4, else goto Step 2 checking the next bit value of the REM register.

Step 4: Add the remainder with the last 16 bits of the input.

Step 5: From the remainder values, the corresponding last 0 – N/3 bits are updated in the sign register.

It is seen from the algorithm that in step 2, the register bit is taken and put onto the sign bit. The

remaining angle is found only when the bit value is 1, thereby reducing the number of subtractions required. Once the calculation for the first (N/3 –1) bits is done, the remainder is added with the last N/3 to N bits of the input and the resultant is given to the LSB's of the sign bit. On an addition, it is noted that there is a possibility of an occurrence of a carry bit, which also should be taken into account and is stored as the $N/3^{th}$ bit in the sign register. Therefore the sign register always consists of N bits as compared to the N–1 input bits. The extra carry bit is placed as the $(N/3 -1)^{th}$ bit in the sign register. Thus the entire N signed bits values representing the rotation are got to the sign register.

Architecture:

From this algorithm, architecture is built that generates the sign bits given any angle. The architecture is built and simulated using Altera Quartus II. The objective of building architecture is to reduce the size such that the number of cells occupied when burnt on an FPGA is minimal. From the algorithm, it is seen that the architecture contains no ROM, as is done the conventional Flat CORDIC architecture. The memory access time is thus reduced in the proposed method. An architecture based on the algorithm is shown in Fig. 1.

The architecture shown is for 24-bit input. Therefore bits 0 – 7 are considered critical bits and the generation of the sign bit for these bits is through a combinational logic. For the remaining 16 bits the sign bits are generated from the adder. Initially, the first 8 bits are extracted and stored in a separate register REM. The remaining angle (16 bits) is stored in another register REM1. Since the number of critical angle bits is 8, eight angle registers are loaded with their respective angle values. The first 8 bits are extracted one after another and passed through a combinational logic and is loaded onto the sign register. Fig 2 shows the detailed working of the combinational logic. Once the first 8 bits are found, the values of registers REM and REM1 are added and the last 16 bits of the sign bits are loaded directly into it. Once the signed bits are found and these are issued to the main CORDIC architecture and the respective trigonometric and logarithmic values are obtained. The details about the hardware simulation used and results obtained from the experimental set-up are discussed in the next section. Thus, the pre-computation of signed digits is performed to build the Flat CORDIC.

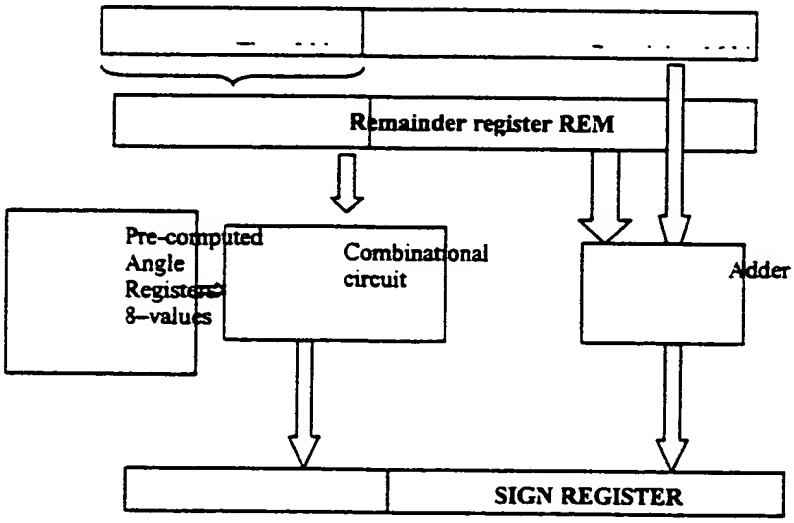
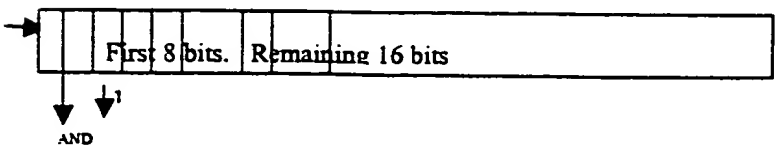


Figure 1: Pre-computation signed digits evaluation architecture



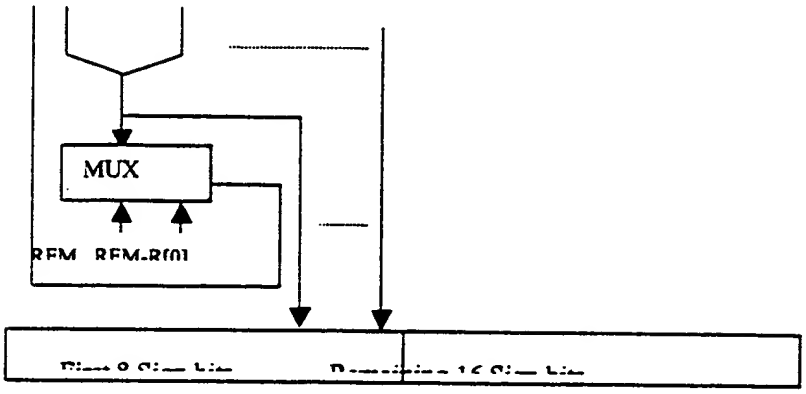


Figure 2: Combinational logic circuit used in the architecture

4. Simulation Results and Discussion

The architecture designed is simulated using Altera Quartus II VHDL. A special case of N=24 is selected to simulate the architecture. For N=24 the first N/3 critical angle is the first 8 bits. A particular angle of 35° is selected as an example. The binary equivalent of this angle in radians is given to the simulation as input. The details about the angles and the results are given in Table 1. The 24-bit angle for the eight different cases is stored in eight different registers and the signed digit calculation for the first eight input bits is done using simple combinational logic and stored in the final sign bit register. The direction of rotation is stored, as 1 or 0, where 1 represents a rotation in counterclockwise direction and 0 represent no rotation. Once the first eight bits are calculated, the remaining angle is added and is stored in the sign register.

The number of clock cycles required is 8 (for the first eight bits) + 1 (for addition). It can also be seen that dividing the input angle to N/3 bits is the optimal way for generating the signed bits, which approximately has a 24-bit precision. Additional clock cycles are saved when computing other trigonometric functions of the input angle. This is because the iterations undergo no rotation during a 0 sign bit. The number of savings in the clock cycle depends on the number of 0's present while computing the sign bits.

With regard to the real estate occupied by the hardware, the prime constituent of the real estate-memory - is not present in this design. This implies that a smaller area is sufficient to accommodate the architecture compared to the conventional Flat CORDIC architecture, which would have required a 256×24-bit ROM for the same specification. From the experimental results it can be seen that the number of logic cells used is 290 operating at a frequency of 3GHz. The FPGA device that is used to build the chip is EP20K100QC208-1, which comes under the family of APEX20K. A comparison of two Flat CORDIC pre-computation architectures is presented in Table 2. It can be seen that the earlier architecture uses 20 times as much as space as compared to the proposed method. This saving in space is compensated to the number of clock cycles used. The proposed architecture uses 9 clock cycles to compute the values compared to 4 clock cycles used by the earlier architecture. But since uni-directional CORDIC is here, the number of iterations required to compute any value is reduced significantly, thereby compensating for the 5 extra clock cycles used for pre-computation.

Table 1. Simulation results for 24-bit input angle

Input Angle	35.0
Input Angle (in radians)	0.610865238
Binary Equivalent of the angle (24 bits)	100111000110000110101010

Value of the first 8 bits	0.609375
Remaining 17 bit value	0.001490238
First 8 sign bit	10100101
Remainder after the first 8 iteration	0.001842437
Total Remaining Angle	0.003332675
Signed digits for the last 16 bits	1101101001101011

Table 2. Comparison of VLSI properties of two architectures

	Proposed architecture	Architecture in [11]
Number of Logic cells	290	6200
Number of clock cycles	9	4

5. Conclusion

A novel algorithm using only one direction of vector rotations to implement the Flat CORDIC architecture has been presented. The unidirectional CORDIC algorithm and the development of the pre-computation of the signed digits for Flat-CORDIC has been described. It has been observed that the proposed technique is efficient with respect to the speed (by reducing the number of iterations) and also the VLSI area (by eliminating the memory) requirement in the implementation of the Flat-CORDIC hardware.

References

[1] J. E. Volder, "The CORDIC trigonometric computing technique," *IEEE Trans. Electronic Computers*, pp. 330-334, Sep. 1959.

[2] J.S. Walther, "A unified algorithm for elementary functions," *Proc. Joint Computer Conf.*, pp. 379-385, 1971.

[3] Y.H. Hu, " CORDIC-based vlsi architecture for digital signal processing." *IEEE Trans. Computers*, vol.42, no.1, pp. 99-102, Jan. 1993.

[4] E. Antelo, J. Villalba, J. D. Bruguera, and E. L. Zapata, "High performance rotation architectures based on the radix-4 CORDIC algorithm," *IEEE Trans. Computers*, vol.46, no.8, pp. 855-870, Aug. 1997.

[5] H. Dawid and H. Meyr, "The differential CORDIC algorithm: constant scale factor redundant implementation without correcting iterations." *IEEE Trans. Computers*, vol. 45, no. 3, pp. 307-318, Mar. 1996.

[6] M. Kuhlmann and K. K. Parhi, "A high-speed CORDIC algorithm and architecture for dsp applications," *IEEE Workshop on Signal Processing Systems (SiPS) Design and Implementation*, Oct. 1999.

[7] Yu Hen Hu and Homer H.M. Chern, "A novel implementation of CORDIC algorithm using backward angle recording (BAR)," *IEEE Trans. Computers*, vol. 45, no. 12, pp. 1370-1378, Dec. 1996.

[8] S. Wang, V. Piluri and E. E. Swartzlander, "Hybrid CORDIC algorithms," *IEEE Trans. Computers*, vol. 46, no. 11, pp. 1202-1207, Nov. 1997.

[9] R. Andraka, "A survey of CORDIC algorithms for FPGAs," *Proceedings of the 1998 ACM/SIGDA sixth international Symposium on Field Programmable Gate Arrays, FPGA '98,* pp. 191-200, Feb. 1998.

[10] L.H. Sibul and A.L. Fogelsanger, "Application of coordinate rotation and function generation," *IEEE Int. Symp. Circuits and Systems*, pp. 821-824, 1984.

[11] B. Gisuthan, T. Srikanthan and K. V. Asari, "Design of an efficient digital architecture for the pre-synthesis of direction of micro-rotations in Flat-CORDIC", *Proc. Second International Conference on Information, Communications & Signal Processing - ICICS'99*, no. 2b2-1, no. 261, pp. 01-05, Dec. 1999.

TRANSMITTAL OF APPEAL BRIEF (Large Entity)	Docket No. ITL.0286US

In Re Application Of: **David K. Vavro et al.**

Serial No.	Filing Date	Examiner	Group Art Unit
09/465,634	December 17, 1999	Tonia Moenske	2183

Invention: **Digital Signal Processor Having A Plurality Of Independent Dedicated Processors**

TO THE ASSISTANT COMMISSIONER FOR PATENTS:

Transmitted herewith in triplicate is the Appeal Brief in this application, with respect to the Notice of Appeal filed on **December 9, 2002.**

The fee for filing this Appeal Brief is: **$320.00**

☒ A check in the amount of the fee is enclosed.

☐ The Commissioner has already been authorized to charge fees in this application to a Deposit Account. A duplicate copy of this sheet is enclosed.

☒ The Commissioner is hereby authorized to charge any fees which may be required, or credit any overpayment to Deposit Account No. **20-1504**
A duplicate copy of this sheet is enclosed.

01/24/2003 DWYATT 00000001 09465634
01 FC:1402 320.00 OP

Signature

Dated: **January 15, 2003**

Timothy N. Trop, Reg. No. 28,994
Trop, Pruner & Hu, P.C.
8554 Katy Freeway, Suite 100
Houston, Texas 77024
(713) 468-8880
(713) 468-8883 (fax)

I certify that this document and fee is being deposited on **01-15-03** with the U.S. Postal Service as first class mail under 37 C.F.R. 1.8 and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Signature of Person Mailing Correspondence

Cynthia L. Hayden

Typed or Printed Name of Person Mailing Correspondence

cc: